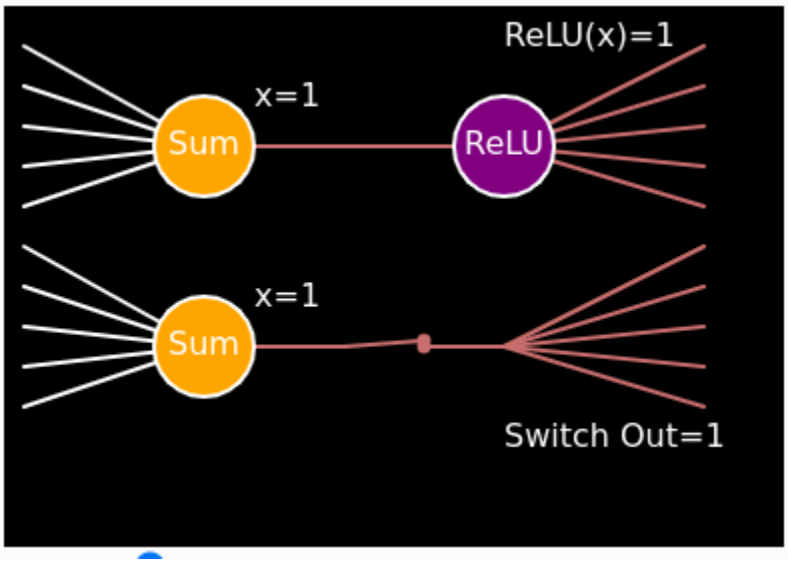
The Secret of BackPropagation

Starting with a typical dense ReLU based neural network and supposing the simplest backpropagation algorithm, stochastic gradient descent (SGD) with batch size 1.
In other words you are training with 1 example <input,target> vector pair at a time and then just backpropagating the training error vector to update each layer in a backward pass through the layers of the network.

The first secret is to understand ReLU is a switch with a switching decision (x>=0)?
That can be difficult to understand or accept because of the historical development of neural networks where switching concepts have never been used and instead the focus was always on activation functions as the non-linear element.
A visual proof that ReLU is equivalent to a switch is shown here:
https://sites.google.com/view/algorithmshortcuts/relu-as-a-switch



A ReLU neural network then is a switched composition of weighted sums. And any time the exact composition is known you can apply linear algebra simplification to condense the composition down to a simple weighted sum in the case of a single neuron or a simple square matrix in the case of an entire neural network.

Secret number 2 is that during the forward pass for a particular input vector, target vector pair all the ReLU switch states become known. You know which weighted sums are connected and which are disconnected. The neural network then can be viewed as a layered composition of linear systems, which itself is linear.
All the stochastic gradient descent algorithm sees is a layered linear system to update. And updating such a system to reduce error is an easy task.

The updates to the weights do however change future switching decisions.
The next time the same particular input target pair is presented for training some of the switching states will have changed.
As training progresses however the number of differently decided switching decisions will reduce. You can say that training with SGD is a form of annealing where you can say the system cools down during the training process.

For a particular input vector to the neural network the sum term of each neuron is equivalent to some simple weighted sum of the input vector (because the switching states in the prior layers have already been decided.) The switching decision at that neuron is directly related to the vector direction of the simple (synthesized) weighted sum, the actual decision boundary being the hyperplane at 90 degrees to the direction of the weighted sum.
Obviously SGD does not know anything about the switching decision boundaries, those just shift around during training.
A question is whether the decision boundaries are actually beneficial or just random.
The question can be answered by replacing the switching decisions with the output of a locality sensitive hash acting on the input to the neural network.
The answer is clear that the learned decision boundaries align in a useful way with the training data and give lower loss even though in a sense they are just learned in an accidental follow along way.